Understanding the Itanium® Architecture

Gautam Doshi
Senior Architect
Intel Corporation

intel®

Intel Labs

Agenda

- Today's Architecture Challenges
- Itanium® Architecture Performance Features
-
-

The Context

- Programmer programs in high level lang.
- Compiler compiles program to machine inst.
- Machine executes these instructions

- High Level Lang = Programmer's vocabulary
- Inst. Set Arch = Compiler's “vocabulary”
  - Architecture determines what the compiler can “express”
  - Architecture determines what the machine must “execute”

Architecture : the compiler's “vocabulary”

Today's Architecture Challenges

- Sequential Semantics of the ISA
- Low Instruction Level Parallelism (ILP)
- Unpredictable Branches, Mem dependencies
- Ever Increasing Memory Latency
- Limited Resources (registers, memory addr)
- Procedure call, Loop pipelining Overhead
- ...

Fundamental challenges abound

Today's Architecture Challenges

Sequential Semantics

- Program = Sequence of instructions
- Implied order of instruction execution
- Potential dependence from inst. to inst.

But ...

- High performance needs parallel execution
- Parallel execution needs independent insts.
- Independent insts must be (re)discovered

Sequentiality inherent in traditional archs

Sequential Semantics ...

Dependent	Independent
add r1 = r2, r3	add r1 = r2, r3
sub r4 = r1, r2	sub r4 = r11, r2
shl r5 = r4, r8	shl r5 = r14, r8

- **Compiler knows the available parallelism**
  - but has no "vocabulary" to express it
- **Hardware must (re)discover parallelism**

Complex hardware needed to (re)extract ILP

Low Inst. Level Parallelism

- Branches: Frequent,   Code blocks: Small
- Limited parallelism within code basic blocks
- Wider machines need <u>more</u> parallel insts.
- Need to exploit ILP across branches
- But some instructions can fault !
- Branches are a barrier to code motion

Limited ILP available within basic blocks

Today's Architecture Challenges

Branch Unpredictability

- **Branches alter the "sequence" of insts.**
- **ILP must be extracted across branches**
- **Branch prediction has its limitations**
  - Not perfect, performance penalty when wrong
  - Need to speculatively execute insts that can fault
    - memory operations (loads), floating-point operations, ...
  - Need to defer exceptions on speculative operations
    - more book keeping overhead hardware

Branches make extracting ILP difficult

Today's Architecture Challenges

Memory Dependencies

- Loads usually at the top of a chain of insts.
- ILP extraction requires moving these loads
- Branches abound and are a barrier
- Stores abound and are also a barrier
  - programming paradigm: Pointers can point anywhere!
- Dynamic disambiguation has its limitations
  - limited in its scope, requires additional hardware
  - adds to code size increase, if done in software

Memory dependencies further limit ILP

Memory Latency

- Has been increasing over time
- Need to distance loads from their uses
- Branches and Stores are barriers
- Cache hierarchy has its limitations
  - Typically small, so limited working set
  - Consumes precious silicon area
  - Helps if there is locality. Hinders, otherwise.
  - Managed asynchronously by hardware

Increasing latency exacerbates ILP need

Today's Architecture Challenges

Resource Constraints

- **Small Register Space**
  - Limits compilers ability to "express" parallelism
  - Creates false dependencies (overcome by renaming)
- **Shared Resources**
  - Condition flags, Control registers, etc.
  - Forces dependencies on otherwise independent insts
- **Floating-Point Resources**
  - Limited performance even in ILP rich applications
  - Data parallel applications need flexible resources

Limited Resources: a <u>fundamental</u> constraint

Today's Architecture Challenges

Procedure Call Overhead

- **Modular programming increasingly used**
  - Programs tend to be call intensive
- **Register space is shared by caller and callee**
- **Call/Returns require register save/restores**
- **Software convention has its limitations**
  - Parameter passing limited
  - Extra saves/restores when not needed

Shared resources create more overhead

Loop Optimization Overhead

- Loops are a common source of good ILP
- Unrolling/Pipelining exploit this ILP
- Prologue/Epilogue cause code expansion
- Unrolling causes more code expansion
- Limits the applicability of these techniques

Loop ILP extraction costs code size

- . . .

- **Complex conditionals**
  - sequential branch execution increases critical path
- **Dynamic resource binding**
  - parallel insts need to be reorganized to fit machine capability
- **(Lack of) Domain specific support**
  - Multimedia: operations repertoire, efficient data-types, …
  - Floating-point: standard compliant, accuracy, speed, …
- **. . .**

And the challenges continue ...

Architecture Challenges

- Sequentiality inherent in traditional architectures
- Complex hardware needed to (re)extract ILP
- Limited ILP available within basic blocks
- Branches make extracting ILP difficult
- Memory dependencies further limit ILP
- Increasing latency exacerbates ILP need
- Limited resources : A fundamental constraint
- Shared resources create more overhead
- Loop ILP extraction costs code size
- And the challenges continue ...

Intel® overcomes these challenges!

Agenda

- Today's Architecture Challenges
- → **Itanium® Architecture Performance Features**
-

Itanium® Architecture Performance Features

- **It's all about Parallelism !**
  - Enabling it
  - Enhancing it
  - Expressing it
  - Exploiting it

… at the proc./thread level for programmer

… at the instruction level for compiler

Enable, Enhance, Express, Exploit - Parallelism

intel®

Intel Labs

Itanium® Architecture Performance Features

- Explicitly Parallel Instruction Semantics
- Predication and Control/Data Speculation
- Massive, Massive Resources (regs, mem)
- Register Stack and its Engine (RSE)
- Memory hierarchy management support
- Software Pipelining Support
- ...

Challenges addressed from the ground up

Explicitly Parallel Semantics

- Program = Sequence of <u>Parallel</u> Inst. Groups
- Implied order of instruction groups
- <u>NO</u> dependence between insts. within group

So ...

- High performance needs parallel execution
- Parallel execution needs <u>in</u>dependent insts.
- Independent instructions explicitly indicated

Parallelism inherent in this architecture

Explicitly Parallel Semantics ...

Dependent	Independent
add r1 = r2, r3 ;;	add r1 = r2, r3 ~~;;~~
sub r4 = r1, r2 ;;	sub r4 = r11, r2 ~~;;~~
shl r5 = r4, r8	shl r5 = r14, r8

- **Compiler knows the available parallelism**
  - and now HAS the “vocabulary” to express it - STOPs (;;)
- **Hardware easily exploits the parallelism**

Frees up hardware for parallel execution

Architecture Challenges

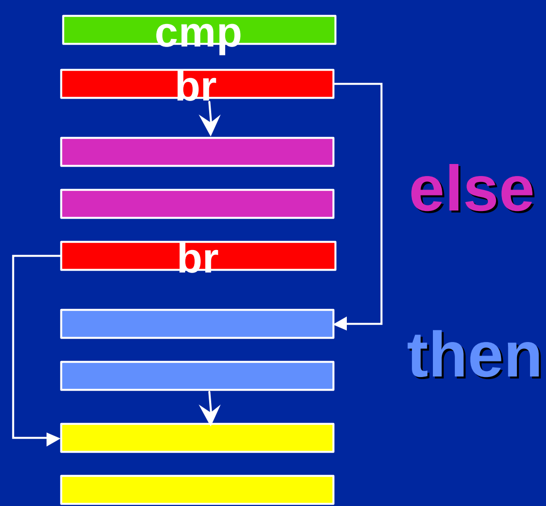
→Sequential Semantics of the ISA

- Low Instruction Level Parallelism (ILP)
- Unpredictable Branches, Mem dependencies
- Ever Increasing Memory Latency
- Limited Resources (registers, memory addr)
- Procedure call, Loop pipelining Overhead
- ...

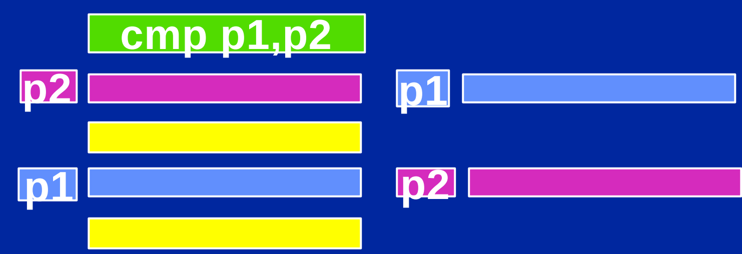
EPIC ISA : Sequential--, Parallel++

Predication

Traditional Arch



Itanium® Architecture



- Control flow to Data flow

Predication removes/reduces branches

Predication ...

- Unpredictable branches removed
  - Misprediction penalties eliminated
- Basic block size increases
  - Compiler has a larger scope to find ILP
- ILP within the basic block increases
  - Both “then” and “else” executed in parallel
- Wider machines are better utilized

Predication enables and enhances ILP

Architecture Challenges

- Sequential Semantics of the ISA
- → Low Instruction Level Parallelism (ILP)
- → Unpredictable Branches, Mem dependencies
- Ever Increasing Memory Latency
- Limited Resources (registers, memory addr)
- Procedure call, Loop pipelining Overhead
- ...

Predication: ILP++, Branches--

Control Speculation

Traditional Arch

instr 1
instr 2
. . .
br

Barrier

ld r1=
use =r1

Itanium® Architecture

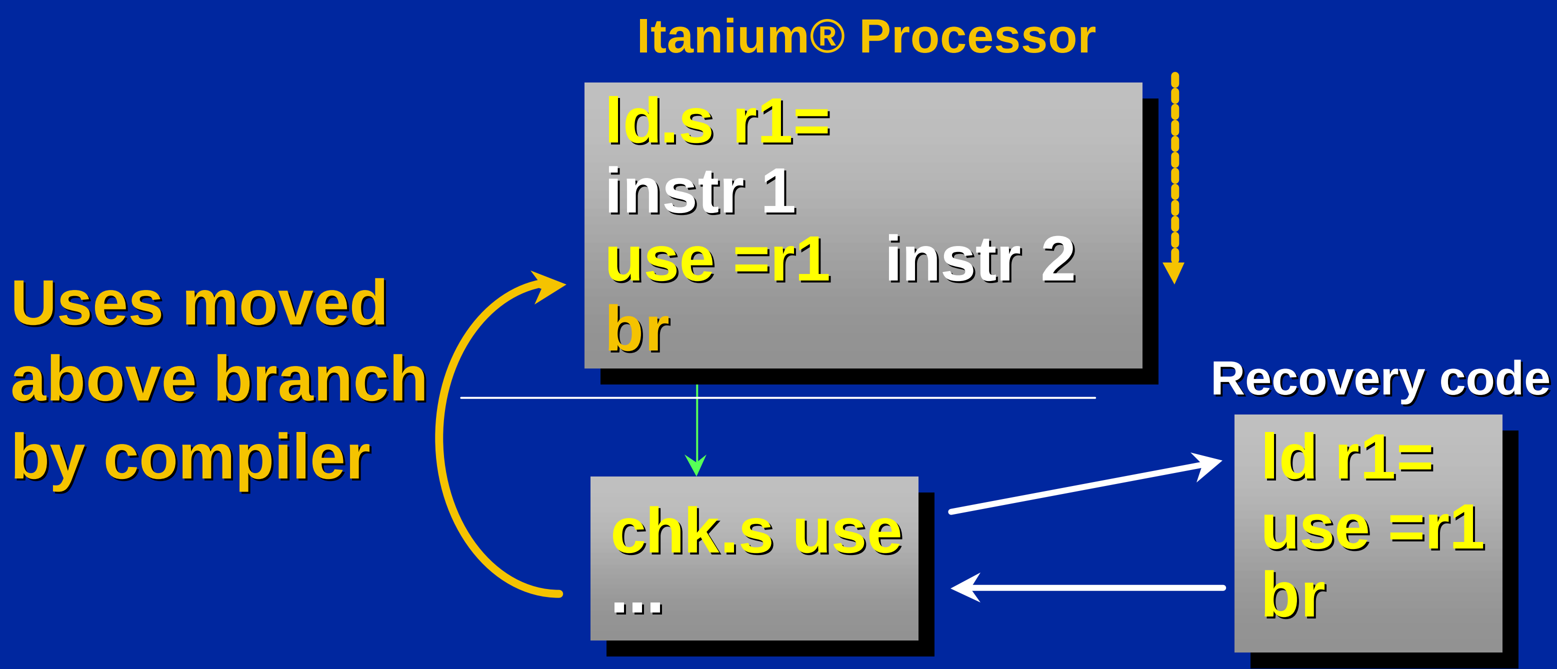
ld.s r1=
instr 1
instr 2
br

chk.s r1
use =r1

Load moved above branch by compiler

Branch barrier broken! Memory latency addressed

Control Speculation ...



- Speculative data uses can also be speculated

Control speculating "uses" further increases ILP

Architecture Challenges

- Sequential Semantics of the ISA
- → Low Instruction Level Parallelism (ILP)
- Unpredictable Branches, Mem dependencies
- → Ever Increasing Memory Latency
- Limited Resources (registers, memory addr)
- Procedure call, Loop pipelining Overhead
- ...

Control Speculation: ILP++, Latency impact--

Data Speculation

Traditional Architectures

instr 1
instr 2
st [?]

Barrier

ld r1=
use =r1

Itanium® Processor

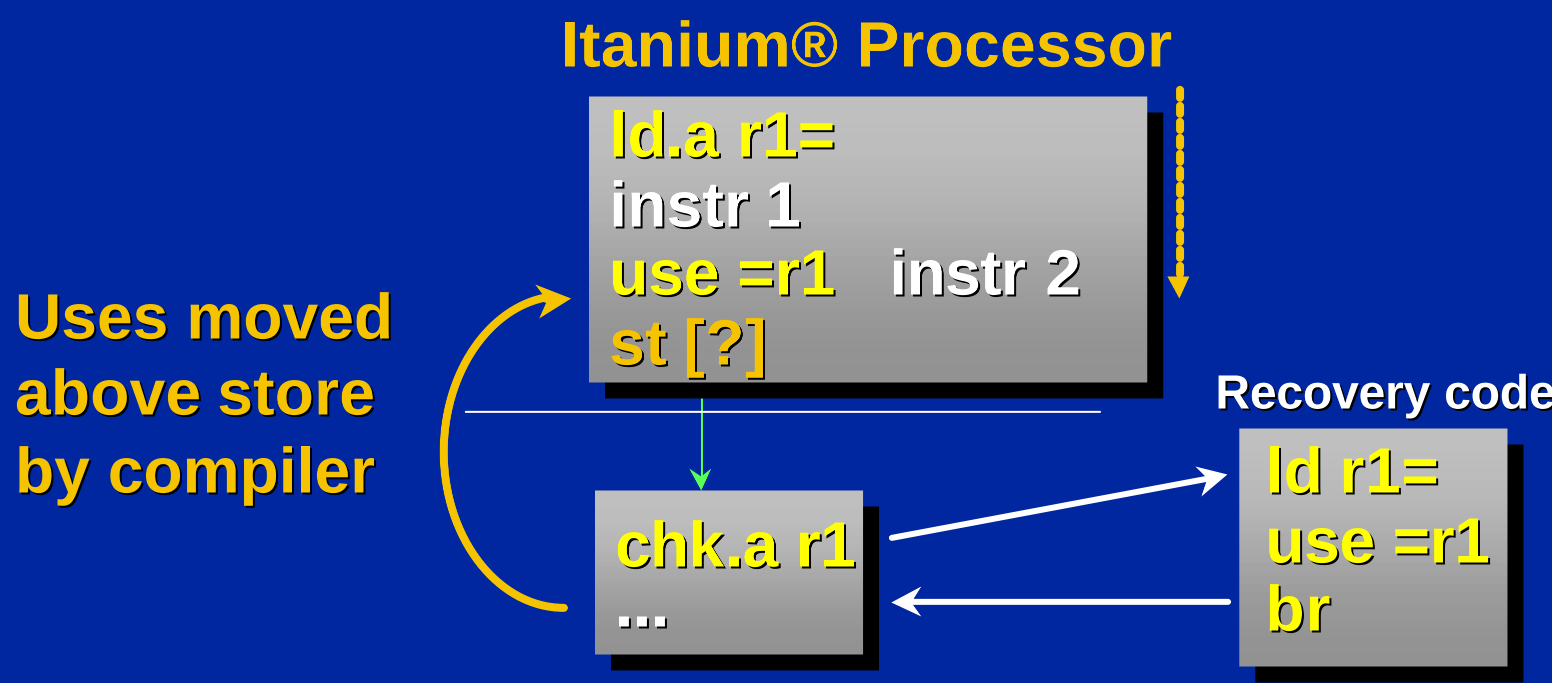
ld.a r1=
instr 1
instr 2
st [?]

ld.c r1
use =r1

Load moved above store by compiler

Store barrier broken! Memory latency addressed

Data Speculation ...



- Speculative data uses can be speculated

Data speculating "uses" further increases ILP

Architecture Challenges

- Sequential Semantics of the ISA

→Low Instruction Level Parallelism (ILP)

→Unpredictable Branches,Mem dependencies

→Ever Increasing Memory Latency

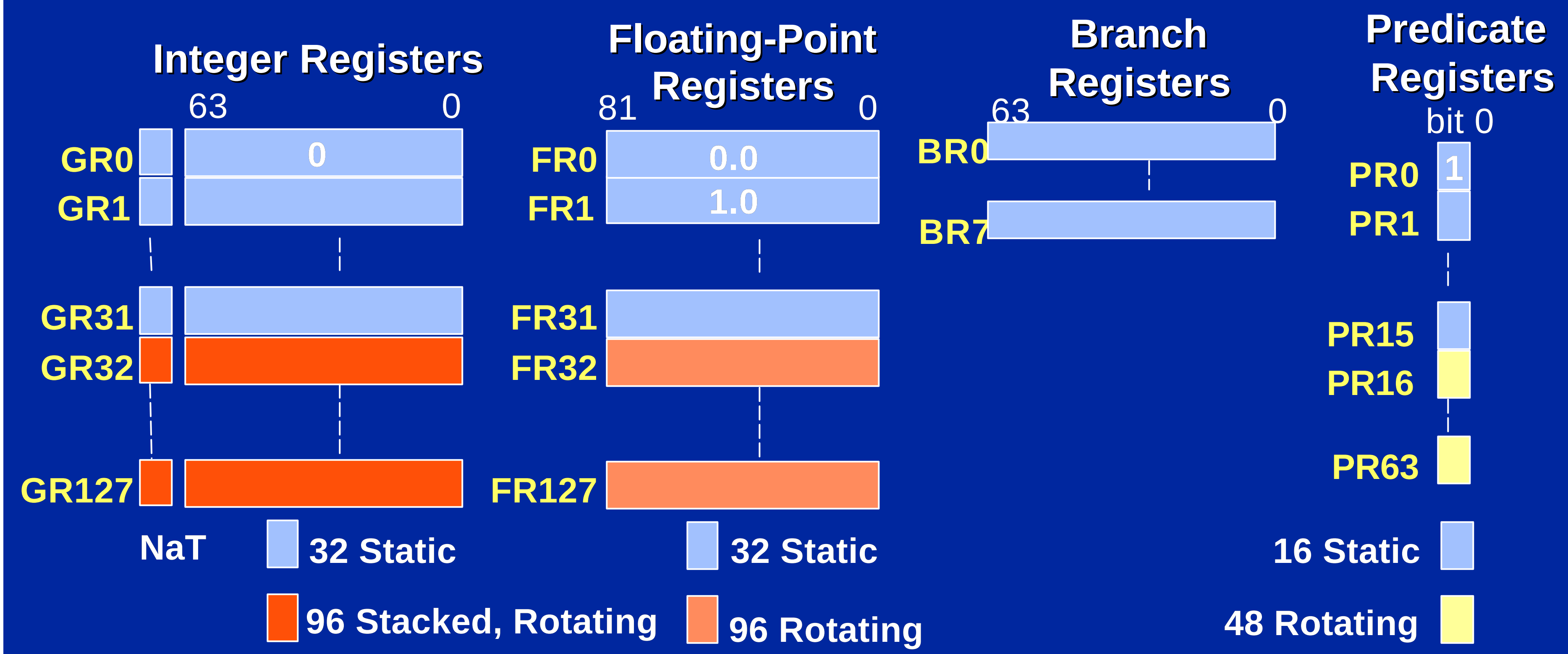
- Limited Resources (registers, memory addr)
- Procedure call, Loop pipelining Overhead
- ...

Data Speculation: ILP++, Latency impact--

intel®

Intel Labs

Massive Execution Resources



An abundance of machine resources

Massive Memory Resources

- 18 BILLION Giga Bytes accessible
  - 2^{64} == 18,446,744,073,709,551,616
- Both 64-bit and 32-bit pointers supported
- Both Little and Big Endian Order supported

An abundance of memory resources

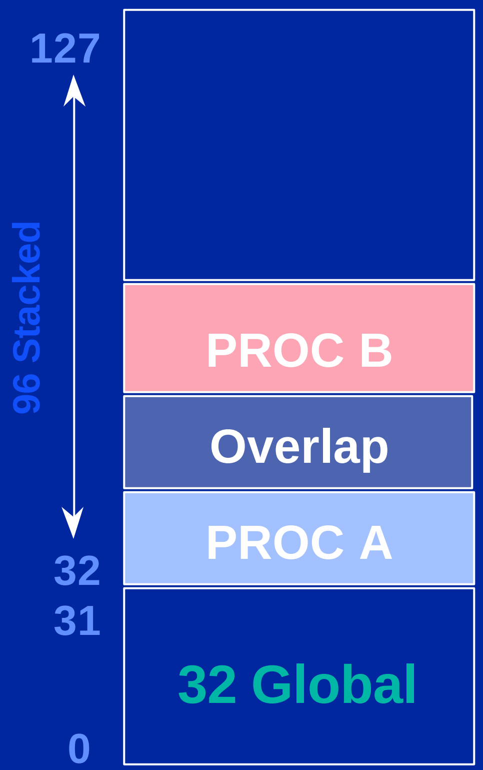
Architecture Challenges

- Sequential Semantics of the ISA
- Low Instruction Level Parallelism (ILP)
- Unpredictable Branches,Mem dependencies
- Ever Increasing Memory Latency
- → Limited Resources (registers, memory addr)
- Procedure call, Loop pipelining Overhead
- ...

Resources: Aid “explicit” parallelism

Register Stack

- GR Stack reduces need for save/restore across calls
- Procedure stack frame of programmable size (0 to 96 regs)
- Mechanism implemented by renaming register addresses

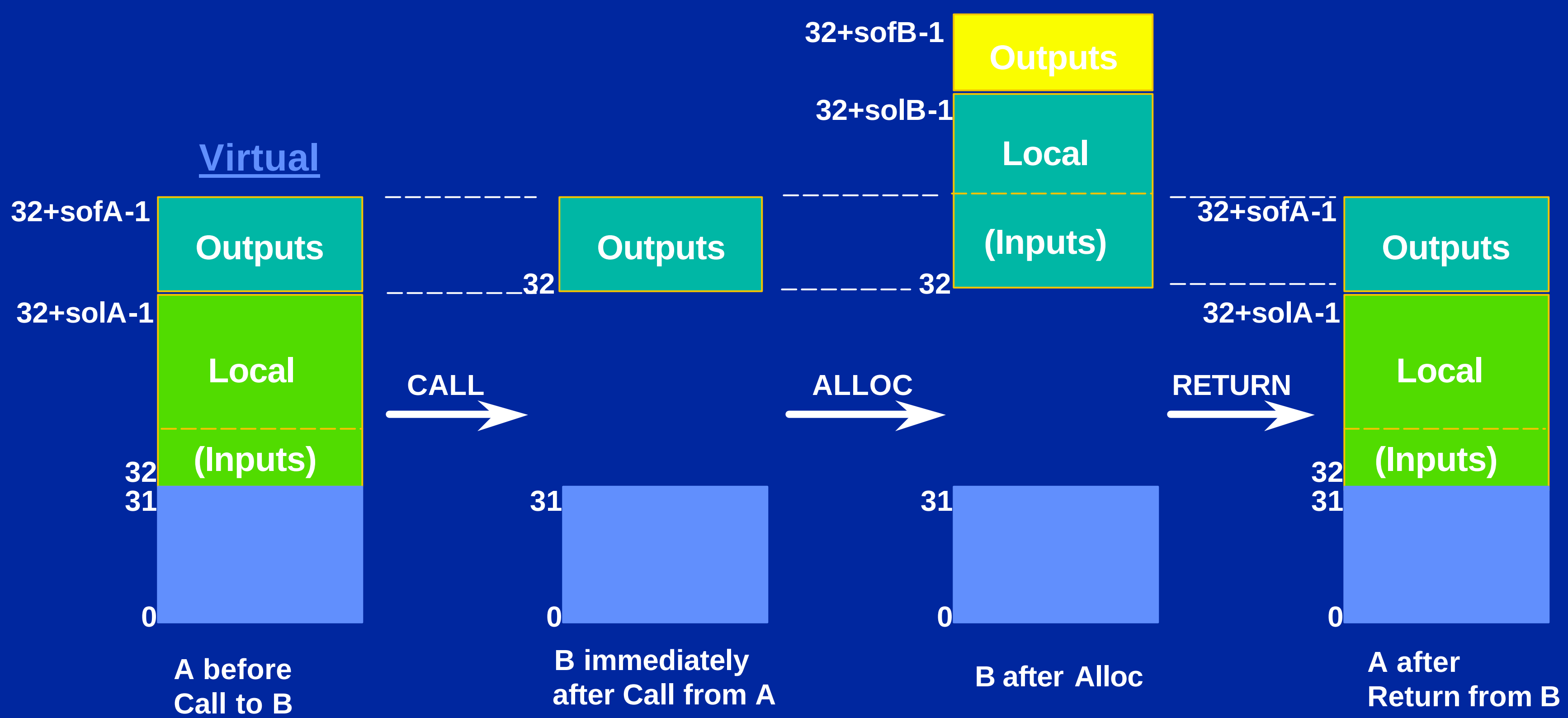


Distinct resources reduce overhead

intel®

Intel Labs

Register Stack



Frame overlap eases parameter passing

Register Stack Engine (RSE)

- **Automatically saves/restores stack registers without software intervention**
  - Provides the illusion of infinite physical registers
    - by mapping to a stack of physical registers in memory
  - Overflow: Alloc needs more registers than available
  - Underflow: Return needs to restore frame saved in memory
- **RSE may be designed to utilize unused memory bandwidth to perform register spill and fill operations in the background**

RSE eliminates stack management overhead

Architecture Challenges

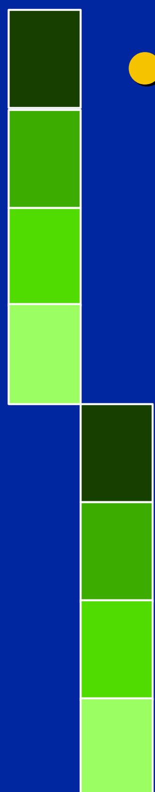
- Sequential Semantics of the ISA
- Low Instruction Level Parallelism (ILP)
- Unpredictable Branches,Mem dependencies
- Ever Increasing Memory Latency
- Limited Resources (registers, memory addr)

→Procedure call, Loop pipelining Overhead

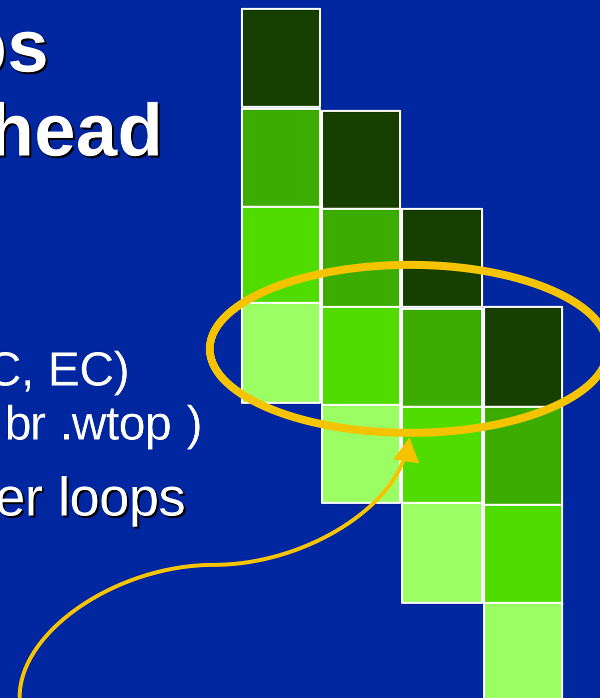
- ...

Reg. Stack: Modular program support

Software Pipelining Support



- **High performance loops without code size overhead**
  - No prologue/epilogue
    - Register rotation (rrb)
    - Predication
    - Loop control registers (LC, EC)
    - Loop branches (br.ctop, br .wtop)
  - Especially valuable for integer loops with small trip counts



Whole loop computation in parallel

Loop support: ILP+++, Overhead---

Architecture Challenges

- Sequential Semantics of the ISA
- Low Instruction Level Parallelism (ILP)
- Unpredictable Branches,Mem dependencies
- Ever Increasing Memory Latency
- Limited Resources (registers, memory addr)

→ Procedure call, Loop pipelining Overhead

- ...

Loop support: Perf. w/o code overhead

Floating-Point Architecture

- **Fused Multiply Add Operation**
  - An efficient core computation unit
- **Abundant Register resources**
  - 128 registers (32 static, 96 rotating)
- **High Precision Data computations**
  - 82-bit unified internal format for all data types
- **Software divide/square-root**
  - High throughput achieved via pipelining

FP: High performance and ___ high precision

Example: Software divide

- **2 dimensional Hydro-dynamics kernel**
  - Livermore FORTRAN Kernel #18
- **Scaling - a common operation**

$ZA_i = N_i/D_i$

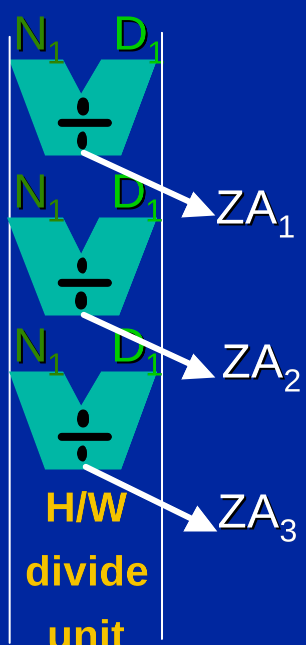
```
   DO 70  k= 2,KN
   DO 70  j= 2,JN
   ZA(j,k)= (ZP(j-1,k+1)+ZQ(j-1,k+1)-ZP(j-1,k)-ZQ(j-1,k))
  .        *(ZR(j,k)+ZR(j-1,k))/(ZM(j-1,k)+ZM(j-1,k+1))
   ZB(j,k)= (ZP(j-1,k)+ZQ(j-1,k)-ZP(j,k)-ZQ(j,k))
  .        *(ZR(j,k)+ZR(j,k-1))/(ZM(j,k)+ZM(j-1,k))
70 CONTINUE
```

Several independent iterations containing divide

Example: Software divide

$$ZA_i = N_i/D_i$$

Traditional Arch

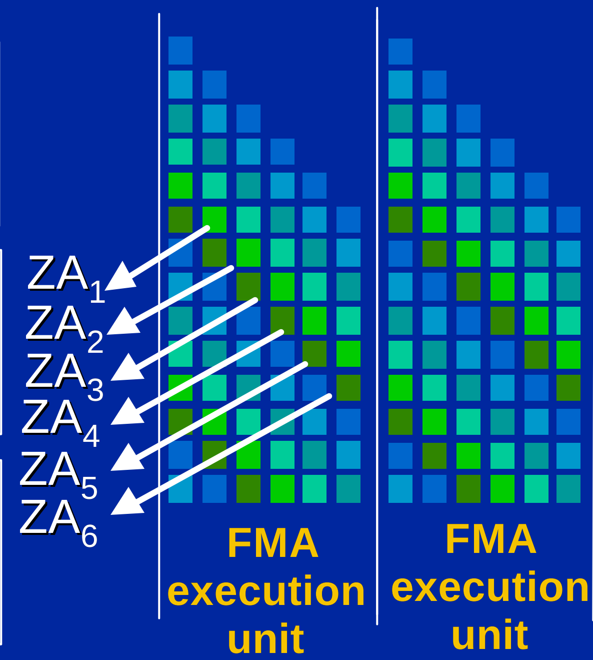


Software divide breaks a single divide into several FMA operations

Slightly greater latency of each divide, but much greater throughput

Performance scales as machine becomes wider and has more FMA execution units

Itanium® Architure



The Software divide provides much greater throughput on FP loops

Intel Labs

...

- **Parallel Compares and Multi-way branches**
  - Control height reduction, branch bandwidth, ...
- **Memory Hierarchy Control**
  - Allocation, De-allocation, Flush, Prefetch (Data/Inst.), …
- **Multimedia Support**
  - Semantically compatible with Intel's MMX™ technology and Streaming SIMD Extension instruction technology
- **Bit/Byte field instructions**
  - Population count, Extract/Deposit, Leading/Trailing zero bytes, ...

And the performance features continue ...

Itanium® Architecture Performance Features

- ◆Parallelism - inherent in the architecture
- ◆Frees up hardware for parallel execution
- ◆Predication reduces branches, enables/enhances ILP
- ◆Control Specn breaks branch barrier, increases ILP
- ◆Data Specn breaks data dependences, increases ILP
- ◆Control and Data Specn address memory latency
- ◆Itanium® provides abundant machine & resources
- ◆Stack/RSE reduces call overhead and management
- ◆Loop support yields performance w/o overhead
- ◆And the performance features continue ...

Beyond traditional RISC capabilities

And YES ...

The Compiler DOES use these powerful architecture features to

- Enable
- Enhance
- Express
- Exploit

the <u>Parallelism</u>